

## 第2章 LR 構文解析 (教科書 p.72)

### 2.1 LR 構文解析の特徴

- パーサーの \_\_\_\_\_ (一方、人手での生成には向かない)  
→ Yacc, Bison などの構文解析器生成系
- 取り扱える文法の範囲が広い
- 演算子順位法と同様に、\_\_\_\_\_ . \_\_\_\_\_ である  
("LR" は Left-to-Right Rightmost derivation に由来する)
  - スタックを用いるのは同じ
  - シフト／還元の判断法がちがう (Bison のプログラム  $\sim .y$  をデバッグするときに判断法の知識が必要になる)  
\_\_\_\_\_ を用いる (ただしスタックの方に!!)  
この DFA は直観的には BNF の右辺のどこまで処理しているか? を表す

### LR 構文解析の例

#### 例 1

$$\begin{array}{l} S' \rightarrow S \$ \\ S \rightarrow ( L ) \\ | \quad x \\ L \rightarrow S \\ | \quad L , S \end{array}$$

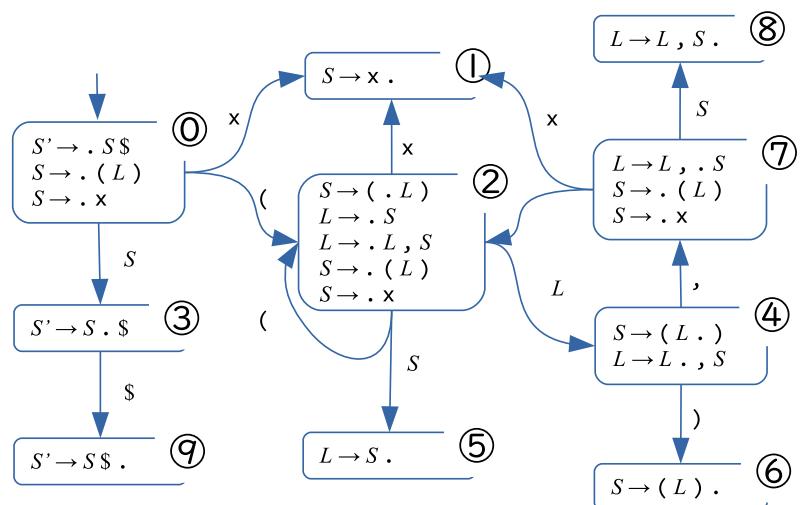
注: LL 法 (後述) は左再帰は扱えないが、LR 法は左再帰の文法のほうが (効率が) 良い。

問 2.1.1 (復習) この BNF の  $S$  から導出される終端記号列の例を導出列とともに 3 つ挙げよ

---

---

この BNF に対応する DFA は以下のようになる (ただし、DFA の作成法は講義の範囲外である)。



输入例: (( \*, \*) , \*) \$

※ … スタックをもう一度 DFA にかける

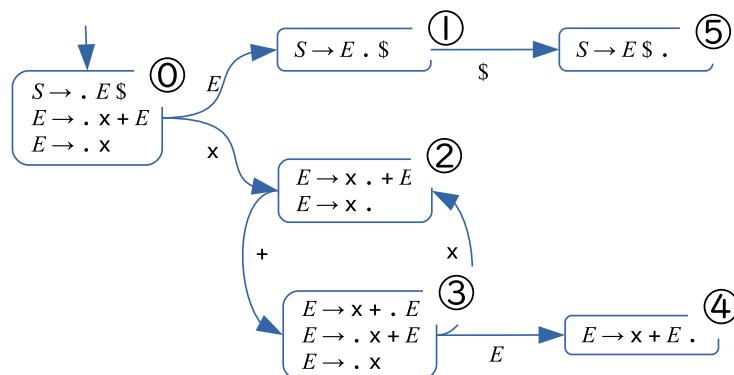
実際にはスタック全体に繰り返し DFA を適用する必要はない。下の表のように

stackに状態も積んでおくと良い

## 例 2

一般にシフト／還元の判定には“先読み”（入力の先頭）も用いる

$$\begin{array}{ll} S \rightarrow E \$ & (1) \\ E \rightarrow x + E & (2) \text{ 右再帰になっている} \\ | & x \end{array}$$



状態②では 先読みが + → shift  
先読みが + 以外 → reduce

## LR 構文解析の階層

先読みの利用の仕方によって、以下のような階層がある

\_\_\_\_\_ > \_\_\_\_\_ > \_\_\_\_\_ > \_\_\_\_\_  
例 2 例 1

それぞれ、以下のような特徴がある

LR(1) … ○ 扱える文法の幅が広い  
× DFA の状態数が多くなりすぎる

LALR(1) … 現実的な選択肢である

Yacc, Bison はこれに基づく  $\leftarrow \text{LL}(1)$  よりは（通常）はるかに広い

SLR(1) … × 扱える文法の幅が狭い

- DFA の状態数は少ない

## LR 構文解析表

例 2 に対して LR 構文解析表は次のようになる（表の作り方は講義の範囲外である）

状態\先読み	x	+	\$	E
①	shift②			goto①
②			shift⑤	
③	reduce(3)	shift③	reduce(3)	
④	shift②			goto④
⑤		reduce(2)	reduce(2)	
	accept	accept	accept	

説明 shift② … shift して状態②へ

reduce(2) … 生成規則(2) を使って reduce

goto① ... 状態①へ

入力例  $x + x + x \$$

## LR 構文解析と曖昧な文法

曖昧な文法に対して無理に LR 構文解析表を作ると \_\_\_\_\_ (conflict) が起こる (つまり、表の 1 つの場所に複数の動作が入る)

### 例 3

$E \rightarrow x \mid E * E \mid E + E$

が起こる。

### 例 $x + x * x$

Yacc (Bison) では演算子の \_\_\_\_\_ · \_\_\_\_\_ を指定して conflict を解消できる

#### 例 4 ( \_\_\_\_\_ , **dangling else**)

$$S \rightarrow \mathbf{if} ( E ) S \\ | \quad \mathbf{if} ( E ) S \mathbf{else} S \\ | \quad \dots$$

例  $\mathbf{if} ( E_1 ) \mathbf{if} ( E_2 ) S_1 \mathbf{else} S_2$  ← これは  
\_\_\_\_\_ と解釈するの  
か？

$\mathbf{if} ( E_1 ) \{ \mathbf{if} ( E_2 ) S_1 \} \mathbf{else} S_2$  と解釈するの  
か？

\_\_\_\_\_ が起こる。 Yacc (Bison) では \_\_\_\_\_ を採用する

#### 例 5 (特殊例の優先)

$$E \rightarrow E^{\wedge} E\_E \mid E^{\wedge} E \mid E\_E \mid \mathbf{id}$$

(優先順位・結合性を与えても) \_\_\_\_\_ が起こる

例  $x^{\wedge} y\_z \$$

Yacc (Bison) では先に書かれている生成規則を優先する

これらの例 3 ~ 5 はよく知られている形だが、このような形以外の conflict は文法を見直す

---

